ISSN 1561-5359. Штучний інтелект, 2018, № 3

УДК 004.41

А.В. Анісімов, О.В. Деревянченко, А.Ю. Хавро
Київський національний університет імені Тараса Шевченка, Україна
пр-т. Глушкова 4Д, м. Київ, 03680

ЗАСТОСУВАННЯ СИСТЕМИ ПАРКС.NET В DOCKER КОНТЕЙНЕРАХ ТА GOOGLE CLOUD PLATFORM ДЛЯ РОЗПОДІЛЕНИХ ХМАРНИХ ОБЧИСЛЕНЬ

A.V. Anisimov, O.V. Derevianchenko, A.U. Khavro
Taras Shevchenko National University of Kyiv, Ukraine
4d, Hlushkova st., Kyiv, 03680

THE APPLYING OF PARCS.NET SYSTEM WITH DOCKER CONTAINERS AND GOOGLE CLOUD PLATFORM FOR DISTRIBUTED CLOUD COMPUTING

У статті запропоновано застосування системи для паралельних розподілених обчислень *ПАРКС.NET* (*ПАРКС* – паралельні асинхронні рекурсивні керуючі системи) та хмарних обчислень. Як технології для хмарних обчислень було використано *Google Cloud Platform*. У роботі надано опис роботи систем *ПАРКС.NET* та можливості її налаштування за допомогою Docker контейнерів для роботи з технологією Google Compute Cloud. Наведено результати тестування систем для вирішення класичної задачі «Пакування рюкзака». Розглянуто можливості цих систем при вирішенні практичних задач паралельного програмування.

Ключові слова: хмарні обчислення, паралельні розподілені обчислення, система *ПАРКС.NET*, Docker, Google Cloud Platform

The article describes the use of the system for parallel distributed computing *PARCS.NET* (*PARCS* - Parallel Asynchronous Recursive Control Systems) in cloud computing. The *Google Cloud Platform* was used as the technology for cloud computing. We provide a description of the PARCS.NET system and its possible settings with Docker containers for the Google Computing Cloud technology. The results of testing systems for solving the classical Knapsack problem are presented. Capabilities of these systems in solving practical problems of parallel programming were considered.

Keywords: cloud computing, parallel distributed computing, *PARCS.NET* system, Docker, Google Cloud Platform

Вступ

Сьогодні хмарні технології допомогають у перенесенні обробки даних з персональних комп’ютерів на потужні сервери мережі Інтернет. З розвитком технологій обробки та збереження інформації та використанням Інтернету обчислювальні ресурси стали дешевші, потужніші та більш розповсюджені. Такий розвиток дозволив реалізацію хмарних обчислень, у яких ресурси (CPU, GPU, RAM та простір на диску) надаються як послуги і можуть бути швидко збільшені, або зменшені, на вимогу користувача, мінімізуючи його затрати. В області комп’ютерного моделювання це відкриває нові можливості для розгортання потужних програмних комплексів із застосуванням хмарних технологій.

Дана стаття присвячена використанню технології ПАРКС (паралельні асинхронні рекурсивні керовані системи) [1-4] для паралельних розподілених обчислень, а саме: системи *ПАРКС.NET* [8] із застосуванням технології контейнеризації для її налаштування у хмарному середовищі. Як технології контейнеризації використано контейнери *Docker* [11] на базі операційної системи Windows. Проведено тестування запуску обчислень за допомогою системи *ПАРКС.NET* в *Docker* контейнерах на хмарній платформі *Google Cloud* [10].

Технологія ПАРКС

ПАРКС – технологія програмування, що являє собою деяку сукупність програмних засобів, які забезпечують процес розробки і реалізації алгоритмів пара-

лельної обробки інформації. Вона базується на концепції керуючого простору (КП) – динамічний граф, який використовується для опису логічної та комунікаційної структури досліджуваної задачі [2].

Основні терміни ПАРКС-технології: точка; програмний канал (ПК); алгоритмічний модуль (АМ).

Структура КП – граф, вершини якого – точки КП, ребра – ПК. До кожної точки КП приписаний АМ, який є процедурою ПАРКС-розширення базової мови [2-4, 6-9]. У нашому випадку ми застосовуємо базову мову програмування C# [5].

Технологія Microsoft .NET Framework

Microsoft .NET Framework – програмна платформа, що дозволяє створювати програми із застосуванням комп'ютерних мереж та паралельних обчислень. Однією з переваг .NET є сумісність бібліотек, написаних різними мовами. Система *ПАРКС.NET* написана мовою C#, але її можна використовувати і для інших мов .NET, наприклад, Visual Basic або F#.

До позитивних якостей технології .NET, що вплинули на її вибір для проектування системи *ПАРКС.NET*, також належать:

- наявність мови C#, яка набуває все більшої популярності завдяки простій об'єктній моделі та постійному розвитку;
- відповідність сучасним вимогам: бібліотека TPL (Task Parallel Library), яка використовується в системі ПАРКС.NET і надає широкий набір можливостей для розробки паралельних програм, підтримка засобів мережевого програмування, забезпечення пересилання даних по комп'ютерній мережі, рефлексія – процес виявлення типів під час виконання та інше [5].

Недоліком платформи .NET Framework є відсутність багатоплатформності. Програми, написані за допомогою платформи .NET Framework, можна запускати тільки на операційній системі Windows. Потенційним шляхом розвитку системи *ПАРКС.NET* є портування її на багатоплатформну платформу .NET Core, що дозволить використовувати систему *ПАРКС.NET* також на операційних системах Linux.

Система ПАРКС.NET

Архітектура системи складається з наступних частин:

1. *Parcs* – основна бібліотека класів системи. Містить всі основні класи, що будуть використовуватись для моделювання паралельних обчислень.
2. *Daemon* – клієнт. Являє собою програму, за допомогою якої будуть проводитись обчислення.
3. *HostServer* – сервер. Має список доступних клієнтів, а також інформацію про поточні задачі та точки. При запуску модуль починає роздавати завдання клієнтам в залежності від кількості ядер процесорів (потужності) та їх завантаженості.
4. *АМ* – окремий проект, що використовує бібліотеку *Parcs* та реалізує інтерфейс *IModule*.
5. *Web* – окремий веб-застосунок для моніторингу поточного стану виконання задач, який також дозволяє завантажувати і запускати свої алгоритмічні модулі.

На всіх машинах, що призначені для обчислень, запускається програма *Daemon*, а початковий АМ та *HostServer* можуть запускатися або на тих же, або на інших машинах. В окремому випадку навіть всі три елементи системи можуть бути запущені в одному місці.

У рамках проведеного дослідження було розроблено новий, кращий для хмарних обчислень алгоритм комунікації клієнта і сервера. Зараз *HostServer* не повинен знати про всі машини, на яких запущений *Daemon*. Натомість, *Daemon* при запуску з'єднується з *HostServer* і передає йому всю необхідну інформацію про себе. Таким чином, ми можемо легко додавати нові обчислювачі до кластера, не вносячи змін в конфігурації *HostServer*.

У створеній реалізації зв'язок проходить у такій послідовності:

1. Спочатку має бути запущений *HostServer*. При запуску він починає очікувати на з'єднання інших компонентів по протоколу TCP.
2. При запуску кожен *Daemon* відправляє на *HostServer* повідомлення про свій запуск, а також таку інформацію про себе, як публічна або приватна IP-адреса, кількість процесорів і результат тестування їх потужності.
3. При запуску модуля *HostServer* розподіляє точки, що створюються в алгоритмічному модулі, по комп'ютерах, на яких виконується програма *Daemon*.
4. Після цього на комп'ютері, де буде створена нова точка, початковий АМ передає exe-файл, у якому міститься інформація про всі АМ, що будуть запущені під час виконання поточної задачі. За допомогою рефлексії *Daemon* створює екземпляр класу АМ і викликає метод Run(). Таким чином запускаються необхідні обчислення.
5. Після завершення обчислень *Daemon* відсилає результат назад на початковий АМ. Після того, як завершилися обчислення на всіх машинах, початковий АМ оброблює всі результати, виводить загальний результат на екран та зберігає його в файл.

Також було додано можливість комунікації компонентів системи через зовнішні (публічні) IP-адреси. Програми *Daemon* і *HostServer* мають параметр: EXTERNAL_LOCAL_IP_ADDRESS, який вказує, за якою IP-адресою інші компоненти будуть звертатися до компонента, у якого вказаний даний параметр.

Технологія контейнеризації Docker

Docker – це технологія контейнеризації, яка дозволяє розробляти, розгортати і запускати програми в так званих контейнерах. *Docker* є одночасно і інструментом розробника, і середовищем виконання. Контейнери містять в собі виконувані файли програми, а також оточення, необхідне програмі для коректної роботи, таким чином надаючи ізоляцію програмам. Контейнери є аналогом віртуальних машин, але є більш легкими в плані затрати ресурсів. На відміну від віртуальних машин, контейнери запускаються на основі одної операційної системи, тоді як кожна віртуальна машина містить у собі окремий образ операційної системи [4].

Основними поняттями в екосистемі *Docker* є образи, контейнери і реєстри. Образ – це статичний опис того, як створити і запустити контейнер, який має свою файлову систему. Образи мають шарову структуру. Кожен образ створюється на базі іншого, наприклад, на основі образу операційної системи Windows. Це дозволяє робити окремі образи невеликого розміру, так як більшість необхідного оточення і функціоналу знаходиться у базовому образі. Образи створюються за допомогою команди *docker build*.

Контейнери запускаються на основі образу. По суті контейнери – це екземпляри образу. Контейнери є оточенням, в якому виконуються програми. Їх можна запускати – *docker run*, зупиняти – *docker stop*, видаляти – *docker rm*.

Ключовим компонентом в екосистемі Docker є реєстри. Реєстр – це сховище образів. Він дозволяє завантажити або скачати образи. Реєстри можуть бути публічними і приватними. Публічним реєстром є *Docker Hub* [12]. Він містить у собі величезний обсяг образів. Також будь-яка людина може створити свій приватний реєстр. Реєстри дозволяють розповсюджувати образи. Завантаження образу в реєстр відбувається за допомогою команди – *docker push,* а скачування – *docker pull*.

Образи *Docker* створюються за допомогою покрокових інструкцій, описаних у файлі Dockerfile. Кожна інструкція створює новий шар. Інструкціями можуть бути такі дії:

- зазначення базового образу (команда FROM);
- запуск команди з командного рядка (команда CMD);

- копіювання файлів в файлову систему контейнера (команда COPY);
- створення змінної оточення (команда ENV);
- відкриття порту для мережевих з'єднань (команда EXPOSE);
- та інші.

Технологія *Docker* спочатку була доступна тільки на Linux-подібних операційних системах. В останні роки також з'явилася підтримка операційної системи *Windows*. Це робить можливим використання технології *Docker* для створення образів системи *ПАРКС.NET*, адже вона на даний момент працює тільки на операційній системі *Windows*. Контейнери, що запущені з образа, створеного на основі образа операційної системи *Windows*, також називаються Windows-контейнерами, тоді як ті, що запущені з образу, створеного на основі образу Linux-подібної операційної системи, називаються Linux-контейнерами.

Однією з ключових переваг технології *Docker* є наявність інструментів для розгортання і управління Docker-контейнерами. Наприклад, *Google Compute Engine*, один з найбільших постачальників хмарних послуг, пропонує створювати віртуальні машини на основі образу, що містить *Docker* і базовий образ: *Windows Server Core 1709* (https://hub.docker.com/r/microsoft/windowsservercore/). Це дозволяє швидко почати використовувати контейнери *Docker*, не витрачаючи час на встановлення інфраструктури. Для користувачів Linux-контейнерів існують навіть кращі можливості, такі як, наприклад, автоматичне розгортання контейнерів на віртуальних машинах. При створенні віртуальної машини користувачу надається можливість вибору образу з *Docker Hub,* контейнер якого запуститься при запуску машини. На жаль, такі можливості на даний момент відсутні для Windows-контейнерів.

У рамках даного дослідження було створено і викладено в реєстр *Docker Hub* такі образи:

1. Parcs *HostServer* (*../parcshostserver*) – образ для програми HostServer.
2. Parcs Daemon (*../parcsdaemon*) – образ для програми Daemon.
3. Parcs Web (*../parcsweb*) – образ для веб-застосунку для моніторингу роботи системи *ПАРКС.NET*.

Всі образи мають тег *windowsservercore-1709*, що означає, що вони створені на основі базового образу *Windows Server Core 1709*. Це дозволяє запускати їх на *Google Cloud Platform*.

Google Compute Engine

Google Compute Engine – веб-сервіс, який надає обчислювальні потужності в хмарі. Сервіс входить в інфраструктуру *Google Cloud Platform*[10].

Основною функцією *Google Compute Engine* є створення, налаштування і керування віртуальними машинами. Віртуальні машини використовують обчислювальну потужність інфраструктури Google. Користувачу надається широкий вибір образів віртуальної машини різних операційних систем: Linux і Windows Server. Також можна налаштувати характеристики операційної системи.

Google Compute Engine має всі стандартні функції, притаманні будь-яким розвинутим *IaaS (Infrastructure as a Service – Інфрастуктура як послуга)* – платформам. Крім створення віртуальних машин, сервіс дозволяє створювати і зберігати образи віртуальних машин. За допомогою даної функції можна створювати нові віртуальні машини на основі існуючих.

Також важливою особливістю є можливість налаштування безпеки та мережевого доступу. Це дозволяє відкрити доступ до віртуальної машини для зовнішнього світу (за замовчуванням він закритий), а також дозволити комунікацію між самими віртуальними машинами.

Зручний веб-інтерфейс *Google Compute Engine* легко дозволяє запускати і зупиняти віртуальні машини, за необхідності, а також робити всі вищеописані дії.

Використання технології Google Compute Engine

Для початку роботи з *Google Compute Cloud* потрібно пройти реєстрацію на сайті: https://cloud.google.com/. Google надає тестовий безкоштовний пакет користування *Google Cloud Platform* на рік (на рахунку $300), хоча і вимагає вказати номер кредитної картки під час реєстрації. Після проходження реєстрації користувачу потрібно створити новий проект, після чого він може починати працювати з сервісами *Google Cloud Platform*. Нас цікавить сервіс *Compute Engine*, що дозволяє використовувати обчислювальні потужності Google, працюючи з віртуальними машинами.

На рис.1 бачимо, що на даний момент створено 5 віртуальних машин, кожна з яких має своє ім'я і внутрішню IP-адресу. Варто зазначити, що при перезапуску машин внутрішній IP залишається сталим, тоді як зовнішній IP може змінюватися. Машинам не присвоєно зовнішній IP, коли вони знаходяться у зупиненому стані.

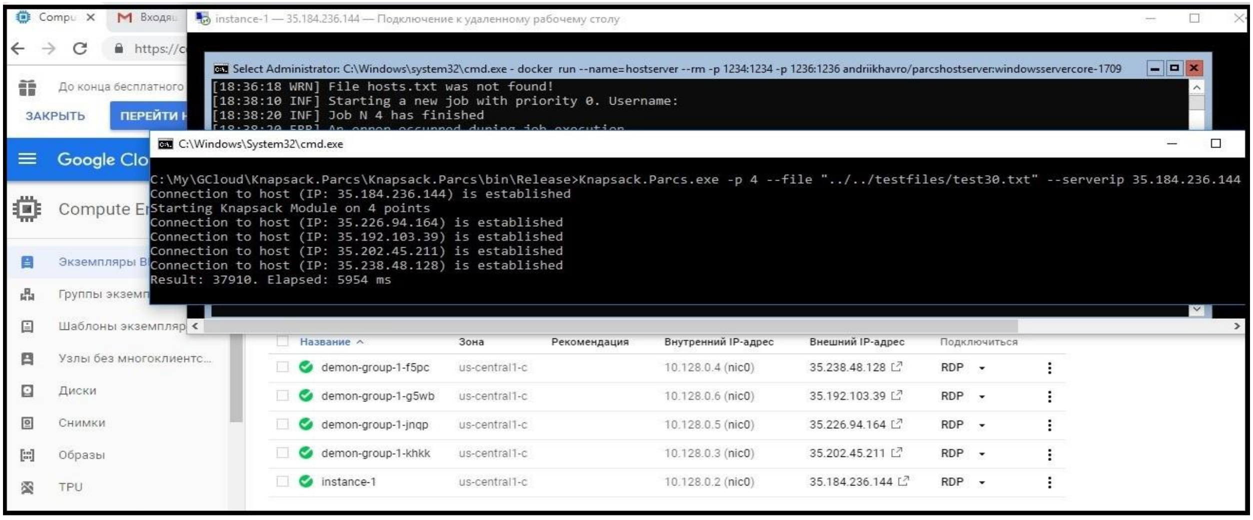


Рис. 1. Запуск *ПАРКС.NET* на *GCP*

Для того щоб створити нову віртуальну машину, потрібно натиснути на кнопку «Create Instance». Далі потрібно вказати назву нової машини, і обрати такі характеристики, як кількість процесорів і оперативної пам'яті. Також потрібно вказати завантажувальний диск. Це може бути образ операційної системи з каталогу Google Cloud, а також образи різних застосунків. Також можна обрати регіон обчислювального центру, в якому будуть проводитися обчислення.

Для тестування оберемо стандартний тип машин з одним процесором (1 vCPU) і 2 GB оперативної пам'яті. Також виберемо регіон europe-west3.

Як образ операційної системи оберемо *Windows Server version 1709 Datacenter Core for Containers*. Цей образ має встановлений *Docker*, а також завантажений базовий образ: *microsoft/windowsservercore:1709*. Це дозволить одразу почати роботу, не витрачаючи час на встановлення залежностей, оскільки цей образ є базовим для образів двох основних компонентів системи *ПАРКС.NET*: *HostServer* і *Daemon*.

Цікавою функцією є можливість вказувати скрипт запуску (Startup script). Цей скрипт виконується при запуску віртуальної машини. За допомогою таких скриптів можна, наприклад, запускати програми, які повинні виконуватися на віртуальних машинах [10].

Під час виконання таких скриптів можна отримати метадані віртуальної машини через Rest API, що надається *Google Compute Engine*. Наприклад, мож-

на отримати інформацію про поточну IP-адресу, і використати цю інформацію при запуску певної програми.

Конфігурація скриптів запуску для віртуальних машин на операційній системі Windows відрізняється від подібних скриптів звичайних віртуальних машин. Для того, щоб додати скрипт мовою Poweshell, який буде запускатися під час запуску машини, треба додати пару «ключ-значення» в секції Custom metadata, де ключем є рядок *windows-startup-script-ps1,* а значенням – сам Poweshell скрипт.

Також віртуальні машини можна створювати з шаблонів. Для того щоб не повторювати процедуру налаштування кожен раз при створенні віртуальної машини, можна зберегти конфігурацію віртуальної машини як шаблон. Для цього потрібно перейти на сторінку *Instance templates*. Створення шаблону є аналогічним створенню віртуальної машини. Описаний вище спосіб створення скриптів запуску також працює з шаблонами [10].

За допомогою шаблонів можна одночасно запустити декілька однакових віртуальних машин. Для цього існують групи віртуальних машин (*Instance groups*). При створенні такої групи можна вибрати шаблон, за яким будуть створені віртуальні машини. Існує можливість налаштувати автоматичне масштабування віртуальних машин (*Auto-scaling*). Якщо ввімкнути таку опцію, *Google Cloud Engine* динамічно створюватиме і видалятиме віртуальні машини відповідно до навантаження. Також можна вибрати параметр, на основі якого буде відбуватися масштабування. Це може бути використання CPU, або кількість HTTP-трафіка. При цьому потрібно вказати мінімальну і максимальну кількість машин у групі. Якщо автоматичне масштабування вимкнуте, потрібно вказати статичну кількість віртуальних машин, які будуть створені у групі. Варто зазначити, що кількість віртуальних машин можна змінювати після створення групи. Таким чином можна додавати або видаляти віртуальні машини з групи.

Наступним кроком є налаштування правил мережевого екрану. Для цього перейдемо на пункт *VPC network* основного меню і перейдемо на сторінку *Firewall rules*. Альтернативно знайти цю сторінку можна через пошук у верхній частині екрана.

Для правильної роботи системи ПАРКС.NET необхідно відкрити мережеві порти, через які спілкуються її компоненти. Для простоти створимо правило для всіх машин у мережі поточного проекту, у якому відкриємо необхідні порти. У нашому випадку: 1234, 1236, 2222.

Налаштування завершено. Далі потрібно підключитися до створеної віртуальної машини. Для цього потрібно встановити пароль *Windows*.

Google Cloud сам створює пароль, його необхідно скопіювати і зберегти. Далі можна завантажити RDP-файл для підключення до віртуальної машини. Відкривши цей файл, потрібно ввести отриманий пароль. Після цього за допомогою захищеного RDP-з'єднання почнеться сеанс роботи з віртуальною машиною.

Віртуальна машина не має графічного інтерфейсу. Користувачу надається тільки консольний інтерфейс, що є достатнім для того, щоб запускати контейнери *Docker*.

За допомогою команди *docker run* запустимо контейнер *HostServer*. Як видно з рис. 2, *Docker* завантажив образ: *../parcshostserver:windowsservercore-1709* і запустив на основі нього контейнер. Важливим є параметр – p, який робить відображення портів контейнера і віртуальної машини, на якій він запущений.

Таким чином, було описано основні способи використання веб-інтерфейсу *Google Compute Engine*.

```
C:\Users\andriy_khavro>docker run --name=hostserver -p 1234:1234 -p 1236:1236 andriikhavro/
parcshostserver:windowsservercore-1709
Unable to find image 'andriikhavro/parcshostserver:windowsservercore-1709' locally
windowsservercore-1709: Pulling from andriikhavro/parcshostserver
5847a47b8593: Already exists
af675e5054a0: Already exists
b4ad692b3ef7: Pull complete
9bd0994a9292: Pull complete
e27a02762f2f: Pull complete
fbbe132bb192: Pull complete
e5e6d83d86f5: Pull complete
Digest: sha256:dded694bdcb687ff8d17301972e1e2613cca093c483a71b95685d35140c47c2d
Status: Downloaded newer image for andriikhavro/parcshostserver:windowsservercore-1709
[17:00:00 WRN] File hosts.txt was not found!
[17:00:00 WRN] Host list is empty!
[17:00:00 INF] Accepting connections from clients, IP: 172.17.155.105, port: 1234
```

Рис. 2. Запуск *Docker* (*HostServer*)

Використання системи ПАРКС.NET на Google Compute Cloud

Для використання системи ПАРКС.NET на Google Compute Cloud необхідно виконати наступні кроки:

1. Створити віртуальну машину на основі образу: *windows-1709-core-for-containers*.
2. Запустити *Docker* контейнер для HostServer на основі образу: *../parcshostserver:windowsservercore-1709*. Також це можна зробити за допомогою скрипту запуску.
3. Налаштувати правила мережевого екрану.
4. Створити шаблон віртуальної машини для *Daemon*. У розділі Custom metadata прописати Powershell скрипт для запуску контейнера *Daemon* з параметром зі змінною оточення: EXTERNAL_LOCAL_IP_ADDRESS. Значення змінної, зовнішню IP-адресу можна отримати за допомогою HTTP запиту до Google API. Також потрібно вказати змінну оточення: PARCS_HOST_SERVER_IP_ADDRESS з IP-адресою машини, на якій запущено *HostServer*. На рис. 3 наведено приклад такого скрипта.
5. За допомогою групи віртуальних машин (Instance groups) створити і запустити необхідну кількість віртуальних машин на основі описаного вище шаблону. На рис. 1 маємо 4 віртуальні машини в групі.
6. Дочекатись, поки всі екземпляри *Daemon* підключаться до *HostServer*. Після цього в консолі *HostServer* з'являться повідомлення про підключення кожного екземпляру *Daemon* рис. 2.

```
$ExternalIp = Invoke-RestMethod
"http://metadata.google.internal/computeMetadata/v1/instance/network-
interfaces/0/access-configs/0/external-ip" -Headers @{"Metadata-Flavor"=
"Google"}

docker run --rm -p 2222:2222 -e
PARCS_HOST_SERVER_IP_ADDRESS=10.156.0.2 -e
EXTERNAL_LOCAL_IP_ADDRESS=$ExternalIp
andriikhavro/parcsdaemon:windowsservercore-1709
```

Рис. 3. Скрипт для запуску parcs-VM

Варто зазначити, що *Google Compute Engine* має обмеження (квоти) на кількість віртуальних процесорів (vCPU) в одному регіоні, а також глобальне обмеження на всі регіони. Це означає, що в межах одного проекту кількість процесорів віртуальних машин має відповідати цим обмеженням. У межах

безкоштовної підписки можна використовувати одночасно не більше 8 віртуальних процесорів, у сумі по всіх регіонах кількість процесорів не більше 24.

Таке обмеження можна обійти за допомогою створення нових проектів. При цьому для кожного проекту потрібно виконати кроки 3-5. Також віртуальні машини кожного проекту будуть знаходитися у власній мережі, тому підключення до *HostServer*, запущеного на віртуальній машині в іншому проекті, необхідно проводити через зовнішню IP-адресу.

Після проведення цих кроків система *ПАРКС.NET* готова до проведення обчислень за допомогою запуску алгоритмічних модулів. За рахунок використання зовнішніх IP-адрес можна запускати алгоритмічні модулі з зовнішньої мережі. Наприклад, ми запустимо нашу тестову задачу з персонального комп'ютера, рис. 1 (у вікні командної строки).

Тестування системи ПАРКС.NET на Google Compute Engine

Для тестування роботи системи було реалізовано алгоритм перебору для розв'язання класичної 0-1 задачі *«Пакування рюкзака»*. Дана задача є NP-повною. Алгоритм не є оптимальним для вирішення даної задачі, але є легким для розпаралелення. Складність даного алгоритму є $O(2^n/p)$, де n – це кількість предметів, а p – кількість процесорів, на яких проводиться обчислення.

Обчислення проводилось на *Google Compute Engine* з використанням до 32 віртуальних машин з одним віртуальним процесором (vCPU). Результати наведені в Таблиці 1.

Таблиця 1. Час (с.) розв'язання задачі для різної кількості VM (p) і предметів (n)

p(VM) \ n	33	34	35	36	37	38	39	40
1	44.2	87.6	174.6	344.4	-	-	-	-
2	22.2	45.7	89.7	180.7	328.3	-	-	-
4	12.2	24.6	47.2	89.8	180.0	328.3	-	-
8	7.5	14.5	25.1	48.5	97.1	184.8	354.4	-
16	6.6	10.2	14.8	27.0	49.9	98.8	185.5	359.4
32	-	11.4	13.6	20.4	32.2	55.6	101.5	197.1

Тести показують, що зі збільшенням кількості машин досягнуто покращення в часі до 17 разів. Також результати підтверджують складність алгоритму $O(2^n/p)$. Видно, що при додаванні одного предмета час виконання збільшується вдвічі. При цьому, час виконання зворотно пропорційний кількості процесорів, на яких виконуються обчислення.

Висновки

У даній роботі було представлено можливість використання системи *ПАРКС.NET* для хмарних обчислень за допомогою технології контейнеризації *Docker*. Було проведено тестування системи на класичній задачі «*Пакування рюкзака*». Також було детально описано можливості *Google Compute Engine*, зокрема створення віртуальних машин, шаблонів віртуальних машин, груп віртуальних машин, налаштування безпеки, що значно спрощує розгортання та налаштування системи *ПАРКС.NET*. При виконанні обчислень на *Google Compute Engine* із застосуванням 32 віртуальних машин було досягнуто покращення у часі в 16-17 разів. При збільшенні розмірності обчислень маємо збільшення ефективності застосування системи *ПАРКС.NET*, що на практиці дає досить непоганий результат.

Потенційним шляхом покращення системи є портування її на платформу

.NET Core, що дозволить використовувати її на машинах з операційними системами сімейства *Unix/Linux*.

У порівнянні з попереднім дослідженням, що проводилося за допомогою обчислювальної потужності *Amazon Elastic Compute Cloud* [8], було досягнуто простіший спосіб розгортання системи *ПАРКС.NET* у хмарі за допомогою технології *Docker* і покращеного алгоритму комунікації *HostServer* та екземплярів *Daemon*. Він дозволяє додавати нові обчислювачі без перезавантаження *HostServer*, що спрощує розгортання і роботу системи.

Також було додано і перевірено можливість роботи системи через зовнішні (публічні) IP-адреси. Це дозволяє об'єднувати через систему обчислювальні комплекси, які знаходяться в різних мережах. Це відкриває нові можливості для створення комбінованих хмарних обчислювальних комплексів більшої потужності.

Література

1. Глушков, В.М., Анисимов, А.В. (1980) Управляющие пространства в асинхронных параллельных вычислениях. *Кибернетика,* №5, С.1-9.
2. Анисимов, А.В., Кулябко, П.П. (1993) Особенности ПАРУС-технологии. *Кибернетика и системный анализ,* №3, С. 128-137.
3. Анисимов, А.В., Деревянченко, А.В. (2005) Система ПАРУС-JAVA для параллельных вычислений на компьютерных сетях. *Кибернетика и системный анализ*, №1, С.25-36. Отримано з: https://link.springer.com/article/10.1007/s10559-005-0037-4
4. Деревянченко, О.В. (2005) Моделювання паралельних програм за допомогою системи ПАРКС-JAVA. *Наукові записки НаУКМА, Комп'ютерні науки.* С. 47-58.
5. Рихтер, Дж. (2012) *CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C#.* 3-е изд., СПб.: Питер, 928 с.
6. Анісимов, А.В., Деревянченко, О.В. (2013) *Паралельне програмування із застосуванням технології ПАРКС-JAVA.* К.: ТОВ «Компанія ВАІТЕ», 78 с.
7. Деревянченко, О.В. (2014) Системи паралельних обчислень на комп'ютерній мережі на базі технології ПАРКС. *Вісник Київського національного університету імені Тараса Шевченка, Серія фіз.-мат. науки*, №2, С. 124-127.
8. Деревянченко, О.В., Хавро, А.Ю. (2015) Застосування системи ПАРКС.NET та Amazon EC2 для хмарних обчислень. *Вісник Київського національного університету імені Тараса Шевченка, Серія фіз.-мат. науки*, №4, С. 111-118.
9. Деревянченко, О.В., Сакевич, Р.Д. (2017) Застосування системи ПАРКС-Java та Google Cloud Platform для хмарних обчислень. *Вісник Київського національного університету імені Тараса Шевченка, Серія фіз.-мат. науки*, №4, С. 69-74.
10. *Google Cloud Platform веб-сервіс.* Отримано з: https://cloud.google.com
11. *Docker – контейнерна платформа*. Отримано з: https://www.doc.docker.com/
12. *Docker Hub – репозиторій.* Отримано з: https://hub.docker.com/

References

1. Glushkov, V.M., Anisimov, A.V. (1980) Upravlyayuschie prostranstva v asinhronnyih parallelnyih vyichisleniyah. *Kibernetika*, #5, S.1-9.
2. Anisimov, A.V., Kulyabko, P.P. (1993) Osobennosti PARUS-tehnologii. *Kibernetika i sistemnyiy analiz,* #3, S. 128-137.
3. Anisimov, A.V., Derevyanchenko, A.V. (2005) Sistema PARUS-JAVA dlya parallelnyih vyichisleniy na kompyuternyih setyah. *Kibernetika i sistemnyiy analiz,* #1, S.25-36. Available from: https://link.springer.com/article/10.1007/s10559-005-0037-4
4. Derevianchenko, O.V. (2005) Modeliuvannia paralelnykh prohram za dopomohoiu systemy PARKS-JAVA. *Naukovi zapysky NaUKMA, Kompiuterni nauky*. S. 47-58.
5. Rihter, Dzh. (2012) CLR via C#. *Programmirovanie na platforme Microsoft .NET Framework 4.0 na yazyike C#.* 3-e izd., SPb.: Piter, 928 s.
6. Anisymov, A.V., Derevianchenko, O.V. (2013) *Paralelne prohramuvannia iz zastosuvanniam tekhnolohii PARKS-JAVA.* K.: TOV «Kompaniia VAITE», 78 s.
7. Derevianchenko, O.V. (2014) Systemy paralelnykh obchyslen na kompiuternii merezhi na bazi tekhnolohii PARKS. *Visnyk Kyivskoho natsionalnoho universytetu imeni Tarasa Shevchenka, Seriia fiz.-mat. nauky*, №2, S. 124-127.
8. Derevianchenko, O.V., Khavro, A.Iu. (2015) Zastosuvannia systemy PARKS.NET ta Amazon EC2 dlia khmarnykh obchyslen. *Visnyk Kyivskoho natsionalnoho universytetu imeni Tarasa Shevchenka, Seriia fiz.-mat. nauky*, №4, S. 111-118.
9. Derevianchenko, O.V., Sakevych, R.D. (2017) Zastosuvannia systemy PARKS-Java ta Google Cloud Platform dlia khmarnykh obchyslen. *Visnyk Kyivskoho natsionalnoho universytetu imeni Tarasa Shevchenka, Seriia fiz.-mat. nauky*, №4, S. 69-74.
10. *Google Cloud Platform - web service*. Available from: https://cloud.google.com
11. *Docker – container platforms*. Available from: https://www.doc.docker.com/
12. *Docker Hub – public repository.* Available from: https://hub.docker.com/

RESUME

A.V. Anisimov, O.V. Derevianchenko, A.U. Khavro

The applying of PARCS.NET system with Docker containers and Google Cloud Platform for distributed cloud computing

The article describes the use of the system for parallel distributed computing *PARCS.NET* (*PARCS* - Parallel Asynchronous Recursive Control Systems) in cloud computing. The *Google Cloud Platform* was used as the technology for cloud computing.

The basic terms of the *PARCS*-technology [2] are point, programming channel (PCh) and algorithmic module (AM). Pattern of control space(CS) is a graph. Top is a point of control space and edge is PChs. The points are connected per PChs. Each point of CS is assigned with the AM. The AM is a procedure PARCS and extension of the basic programming languages [2-4, 6-9] (now - C# [5]). AM are executed in parallel. In practice this means that in a computer network one computer (*HostServer*) is selected. This computer is accessible to all others (*Daemon*). It generates a queue of all tasks. Tasks from the queue are sent for computing. Each computer (VM) obtains a task from the queue and executes it. After finishing computing the active task it sends the results to the *HostServer* and selects the new task. The model is designed for distributed network.

We provide a description of the *PARCS.NET* system and its possible settings with Docker containers for the Google Computing Cloud technology.

The results of testing systems for solving the Knapsack problem are presented. Capabilities of these systems in solving practical problems of parallel programming were considered.

Надійшла до редакції 24.10.2018